Dynamically Self-Reconfigurable Machine Learning Structure for FPGA Implementation

Janusz Starzyk and Yongtao Guo
School of Electrical Engineering and Computer Science
Ohio University, Athens, OH 45701, U.S.A.
{starzyk, gyt}@bobcat.ent.ohiou.edu

ABSTRACT

In this paper, we describe organization of a machine learning system based on dynamically reconfigurable architecture and self-organization. This system learns typical neural network tasks using self-organizing learning array algorithm described elsewhere. To develop this system, we adopt hardware-software codesign approach based on combining an array of VIRTEX XCV1000 FPGAs with custom software – Matlab/C++. The prototype structure is divided into hardware architecture, software programs and their interface. Hardware architecture dynamically implements the neurons training and voting. Software programs implement control of database and system level management, and are interfaced with hardware via PCI bus using developed C++ dynamic libraries and interface logic.

KEYWORDS

Dynamic reconfiguration, self reconfiguration, hardware-software codesign, VHDL, FPGA.

1 INTRODUCTION

Field programmable gate arrays (FPGAs) have the ability to be dynamically reconfigured either completely or partially (e.g. Xilinx XC6200 FPGA). However, this run-time reconfiguration is usually performed by software programs that decide when and how to reprogram the FPGAs [1]. Configuration data can be stored in a configuration memory and can be utilized to reconfigure the FPGAs within one or two clock cycles. Several case studies have proven the feasibility for these methods, especially in wireless communication application area [2]. However, both the software programs and the precompiled circuit configuration bits have limited abilities to handle machine learning because they use configuration from a limited, prepared ahead of time selections.

Self-organization is important in artificial neural networks (ANNs) and machine learning. It can help machine to display some intelligence and can be implemented using dynamical reconfiguration. A self-organizing learning algorithm that combined neural networks and information theory was presented in [3]. The algorithm was simulated on standard benchmarks and proved to be advantageous [3][4] over many existing machine learning methods. Based on this algorithm, we propose a self-organizing learning array (SOLAR) system which is different from classical ANNs in the way it is organized and how it learns. SOLAR is a parallel processing hardware with dynamical reconfigurability derived from its self-organizing structure. This structure comes with numerous processing components (neurons) and relatively sparse interconnections between them. While classical ANNs have cubic relationship between wiring area and the number of neurons, SOLAR's interconnection area grows almost linearly with the number of neurons. Neurons have simple and identical structure which supports dynamical self-reconfiguration.

To implement SOLAR, we adopt FPGA as the hardware platform since custom VLSI hardware requires significant non-recurring engineering cost. Considering that SOLAR performs intensive data processing, we adopt hardware-software codesign approach [5] to develop its structures. The approach is based on combining off-the-shelf hardware components – VIRTEX XCV1000 FPGAs with software – Matlab and C++.

The rest of this paper is organized as follows. In Section 2, self-organizing learning array including its working principle and hardware architecture is introduced. Section 3 deals with the HW/SW codesign and co-simulation of SOLAR. A summary is given in Section 4.

2 SELF-ORGANIZING LEARNING ARRAY

Solar Principle

Let $S = \{(v, c)\}$ be a training set of N vectors, where $v \in \Re^n$ is a feature vector and $c \in Z$ is its class label from an index set Z. A classifier is a mapping $C : \Re^n \to Z$, which assigns a class label in Z to each vector in $\Re^n$. A training pair $(v, c) \in S$ is misclassified if $C(v) \neq c$. The performance measure of the classifier is the probability of error, i.e. the fraction of the training set that it misclassifies. Our goal is to minimize this misclassification probability and it is achieved through simple threshold searching based on maximum information index, which is calculated from estimated subspace probability and local class probabilities. In SOLAR structure, we estimate the probability distribution of the training data and calculate the system entropy. We use entropy based information index to

evolve the neural network structure in the learning phase.

SOLAR is implemented as a feed forward structure. It has a pre-wired organization that contains a number of identical processing neurons, which are pseudo-randomly connected to the input nodes and other neurons. An entropy-based evaluator is utilized to select a proper operation from different neuron's functions and chose input selection for each neuron. The set of training data is searched sequentially class by class to establish the optimum thresholds, which best separate the signals of various training classes. The quality of the partition is measured by entropy based information index defined by:

$$I = 1 - \frac{\Delta E}{E_{\max}}$$

where $$\Delta E = -\sum_s \sum_c P_{sc} \log(P_{sc}) + \sum_s P_s \log(P_s)$$

and $$E_{\max} = -\sum_c P_c \log(P_c)$$

here, P_c, P_s, P_{sc} represent the probabilities of each class, attribute probability, and joint probability, respectively. The summation is performed with respect to all classes and subspaces.

The information index should be maximized to provide an optimum separation of the input training data. The neuron learning is realized by maximizing information content of all the neurons. When the calculated value of the information index equals to 1, the problem at hand is solved completely, i.e. the training data is correctly classified by SOLAR. If a neuron reaches specified information index threshold, the neuron becomes a voting neuron. Several voting neurons are weighted together to solve a given problem. As reported in [3][4], SOLAR performed well in simulation comparing to many specialized machine learning algorithms and outperformed all ANNs.

Solar Architecture

In SOLAR, we adopted feed forward network structure for its stability and fast learning. SOLAR architecutures, as shown in Fig. 1, are based on the proposed organization of dynamically reconfigurable architecure – DREAM [6]. It is implemented as an array of identical processing units (neurons) with switchable routing channels and programmable functionality. They can be either self reconfigured or dynamically reconfigured by the configuration memory units (CMU) that serve four neurons per CMU. This architecture follows organization of DREAM with this exception that dynamic reconfiguration, initialized by configuration control and defined outside the chip, may be replaced by dynamic self-reconfiguration which is data driven and is local to each neuron. The neuron interconnection is implemented by both unidirectional configurable switch units (CSUs) and bidirectional routing units (BRUs). CSU has a butterfly-like configurable switch structure and can implement pseudo-random input/output connections. BRU is a bidirectional cross connection unit which can assist CSU to implement bidirectional signal transfer in the vertial direction. By combining CSU and BRU, the signal can be passed to either a local neuron or any remote neuron in a forward layer. There are two types of connections in every neuron - local connections with higher connection probabilities, and remote connections with smaller probilities. In SOLAR, initial connections are pseudo-randomly defined by control bits generated by a linear feedback shift register. This pseudo-random interconnection approach applies to both the neuron's input signals as well as its control signal that defines the learning subspace for each neuron. Because of the identical neuron structure and rich and flexible routing resources, SOLAR architecuture expandable to improve its learning ability with numerous neurons.

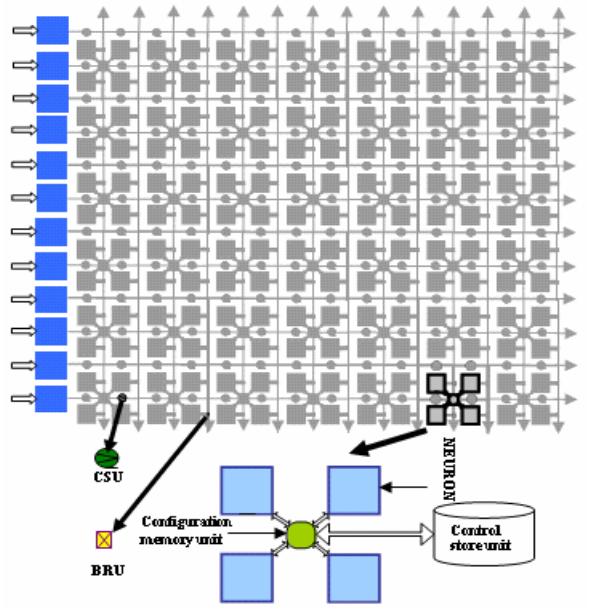

Figure 1. SOLAR organization structures

3 HARDWARE-SOFTWARE CODESIGN

Our ultimate objective is to design SOLAR architecture based on hundreds of high-end FPGA chips to form a 3D learning machine. Currently, we are prototyping SOLAR onto an array of VIRTEX FPGA chips on the demo board [7] which supports up to four VIRTEX FPGAs to verify its hardware implementation in a real environment. Since SOLAR is based on a supervised training algorithm, the system contains two stages – training and voting. After training, SOLAR system is dynamically reconfigured into a voting stage utilizing the trained results. These two stages work at distinct time as shown in Fig. 2. Reconfiguration from training to voting is performed dynamically and is transparent to the system. In final implementation, learning and voting will not be separated in time, and dynamic reconfiguration will be replaced by self-reconfiguration. In both training and voting stages, we model software using Matlab and C++, we model

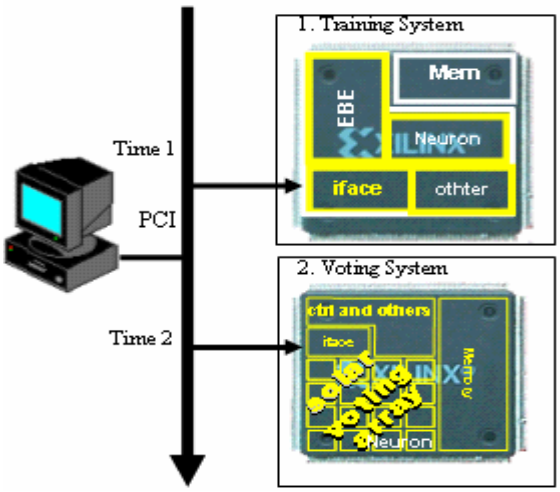

Figure 2. Two stages for SOLAR

hardware using synthesizable VHDL building blocks, and we model HW/SW interface using dynamical link libraries plus PCI core and other interface logic. So, we can decompose the prototyping system into three parts:

- CPU and memory (software): System initialization, organization and management are implemented by Matlab programming. Time-critical loops and recursive applications are implemented in C++.
- FPGA (hardware): Training and voting architectures are dynamically configured onto the chip at distinct times. Voting architecture depends on the learned results from training. This hardware prototype adopts system level organization of DREAM [6] with modification of the routing organization.
- Interface: It contains PCI core and interface logic in the hardware part, and C++ dynamical link libraries called from Matlab console in the software part.

Software Organization

Software component is a significant part of the system prototyping. It requires a significant effort to arrive at a usable, tightly integrated software solution for SOLAR, since it has to handle the system synchronization and data received from hardware. Most of the software models are implemented in Matlab due to its programming simplicity compared to C++. However, Matlab is not very efficient in some time-hungry operations including loop and recursive function calls. Such functions are implemented by C++ and are called by Matlab through dynamical link libraries. Many functions that define the system behavior are organized hierarchically as shown in Fig. 3. The higher level functions encapsulate the functions described on the lower level. The highest level software function implements the system organization and management. Training has more hierarchical levels than voting. This hierarchical organization helps us to communicate with the appropriate level of hardware during both stages.

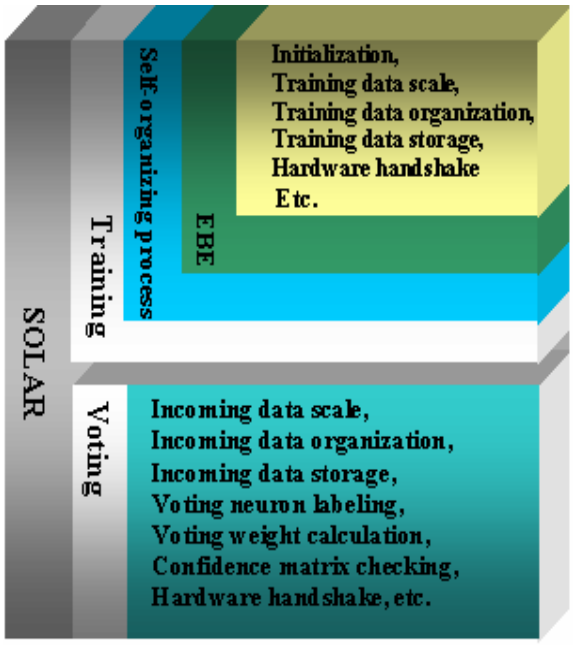

Figure 3. A hierarchical structure of system modelling

Hardware Architecture

Hardware is designed using synthesizable structural VHDL and models the SOLAR architecture as shown in Fig. 4. Data fed from the PCI interface represents the neurons' learning space. Main control module receives data via DMA transfer and stores them into memory "MEM1". These memory modules are implemented by block memories in VIRTEX FPGA chip up to 32x2K bits for the prototype chip. The storage space is sufficient for neurons learning data collected from a standard benchmark to verify the hardware implementation with software. Once the software triggers

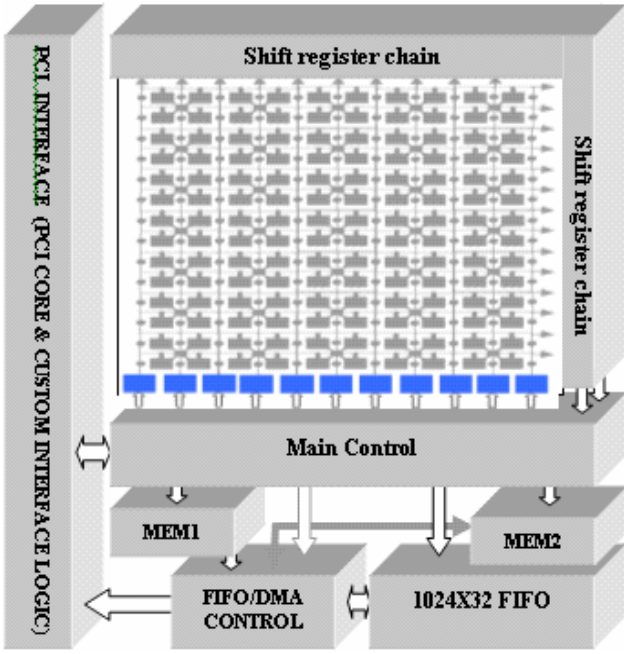

Figure 4. Simple SOLAR architecture prototype

the execution for neurons' training, the main control fetches data from memory 1 and sends them to the SOLAR architecture. At the same time other information including searching step for EBE modules and current information index values representing the learned results, are sent. Data pass through the SOLAR architecture above the main control module shown in Fig.4, and then current learned data in a new subspace is shifted out via shift register chain to memory 2. If the information index corresponding to the current threshold is higher than its previous maximum, data in memory 2 is send out to FIFO controlled by the main controller. The process is repeated until all thresholds are scanned. Finally, the optimal thresholds and corresponding information indexes are stored in dynamic configuration registers. These optimal parameters are read back through rapid DMA transfer requested by the system level functions (software) for storage and further processing. Both training and voting utilize similar prototyping structure based on the proposed SOLAR architecture for neuron's organization.

Hardware-software Interface

In HW/SW interface, the PCI core is used and the code/decode logic and control finite state machine are developed to help communicate with other hardware modules. In software, instead of simply reading and writing hard-coded memory locations to access FPGAs, a dedicated set of routines including control and data I/O functions are developed in C++ DLL to facilitate these operations. The control functions contain opening/closing the chip, sending/receiving control signals, locking and synchronization the FPGA, etc. Data I/O functions contain fetching register data from the chip, DMA transferring, etc. This interface links system organization and management (software) with neuron's self-organizing learning architecture (hardware).

Hardware-software co-verification

Co-verification refers to the process of determining that a HW/SW design is correct. Based on the correct simulations, we prototype SOLAR in a real hardware-software environment – software runs on PC and hardware is configured to the VIRTEX FPGA chip on the demo board [7] which supports up to four VIRTEX FPGA chips. Fig. 5 shows a learned result of one neuron in the array after the real-time verification. For this particular neuron, the selected input subspace contains the raw input data as shown in the upper part of Fig. 5 and one dimensional learned output subspace of the neuron is shown in the lower part of Fig. 5. Data from which the diagram is produced is obtained through the FPGA interface. The figure is generated by Matlab to utilize its strong plotting ability. Matlab simulation produces a very similar result with this real-time hardware/software verification, except minor differences in numerical values for threshold and information index.

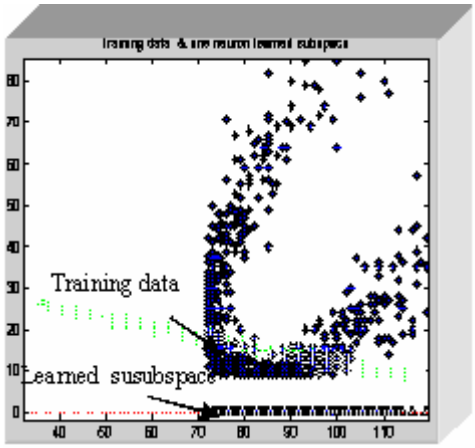

Figure 5. Training data & one neuron's learned subspace

4 SUMMARY

Dynamically reconfigurable architectures can be quickly reconfigured by reading pre-stored configuration bits from memory. SOLAR architecture based on a simple and regular neuron array is similar to these structures. Its reconfiguration can be achieved either from outside chip using CMU presented in DREAM architecture [6] or internally by self-reconfiguration. This way SOLAR is a new class of learning machines and at the same time a new type of reconfigurable hardware.

5 REFERENCE

[1] J. Hogg, "A dynamic hardware generation mechanism," Int. Conf. Designing Correct Circuits, Springer Verlag, 1996.

[2] R. W. Hartenstein, J. Becker et al. "A novel machine paradigm to accelerate scientific computing," Special issue on Scientific Computing of Computer Science and Informatics Journal, Computer Society of India, 1996.

[3] J. A. Starzyk and Z. Zhu, "Software simulation of a self-organizing learning array system". The 6th IASTED Int. Conf. Artificial Intelligence & Soft Comp (ASC 2002), Canada.

[4] J. A. Starzyk and T-H. Liu, "Design of self-organizing learning array," IEEE Int. Symp. on Circuits and Systems (ISCAS), Bangkok, Thailand, May 2003.

[5] Amer Baghdadi, "Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems," IEEE Trans. Software Engineering, Vol. 28, No. 9, pp 822-831.

[6] J. Becker, A. Alsolaim, M. Glesner, and J. Starzyk, "A parallel dynamically reconfigurable architecture for flexible aplication-tailored hardware/software systems in future mobile communication," The Journal of Supercomputing, Erratum Vol. 23, 132, 2002.

[7] Nallatech Ltd, "Ballynuey 2 VIRTEX PCI card user guide," 1993-1999.